Senior Service College FELLOWSHIP PROJECT

ACHILLES' HEAD: UNDERSTANDING THE CHALLENGES IN IMPLEMENTING DEPENDABLE AND SECURE DEEPLY NETWORKED MILITARY EMBEDDED SYSTEMS

BY

LIEUTENANT COLONEL DAVID K. SARJI
United States Army National Guard

USAWC CLASS OF 2008

U.S. Army War College, Carlisle Barracks, PA 17013-5050

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
21-04-2008	Civilian Research Paper	

4. TITLE AND SUBTITLE

Achilles' Head: Understanding the Challenges in Implementing Dependable and Secure Deeply Networked Military Embedded Systems

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

LTC David K. Sarji, ARNG

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Carnegie Mellon University
4720 Forbes Avenue
CIC, Room 2106
Pittsburgh, PA 15213

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Gene Hambrick
Carnegie Melon University
4720 Forbes Avenue, CIC Room 2106
Pittsburgh, PA 15213

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION A: UNLIMITED

13. SUPPLEMENTARY NOTES

The views of the academic research paper are those of the author and do not reflect the official policy of the U.S. Government, the Department of Defense, or any of its agencies.

14. ABSTRACT.

Embedded computing is an essential part of our military capacity. Experience with traditional information technology (IT) systems has made it abundantly that computing systems are subject to attacks, and that including security in the design process is a critical component in the development of new systems.

However, developing secure embedded systems is not a simple matter of transferring security techniques from traditional information technology environments. Due to the strenuous operational environment, frequent interaction with the physical world, and software application domain of embedded computer systems, traditional software and enterprise approaches to achieving dependability and security are only partially effective in securing embedded systems.

This paper will summarize basic concepts of traditional systems dependability and security, and their applicability to embedded systems. Constraints inherent in embedded systems will be discussed and threats to dependability and security summarized. The conceptual gap between regarding security as merely the process of securing communication channels and developing truly secure, deeply networked, embedded systems resistant to all manner of attacks will be explored. Last will be a brief discussion of current best practices for attempting to achieve the appropriate level of dependability in embedded systems, up to and including ultra-reliable systems.

15. SUBJECT TERMS

Embedded Systems, Security, Computing Systems

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			**19b. TELEPHONE NUMBER** *(include area code)*
UNCLASSIFED	UNCLASSIFED	UNCLASSIFED	UNLIMITED	39	

USAWC CIVILIAN RESEARCH PROJECT

ACHILLES' HEAD: UNDERSTANDING THE CHALLENGES IN IMPLEMENTING DEPENDABLE AND SECURE DEEPLY NETWORKED MILITARY EMBEDDED SYSTEMS

by

Lieutenant Colonel David K. Sarji
United States Army National Guard

Gene Hambrick
Project Adviser

Disclaimer

This CRP is submitted in partial fulfillment of the requirements of the U.S. Army War College Fellowship Program.

The views expressed in this student civilian research project are those of the author and do not reflect the official policy or position of the National Guard Bureau, the Department of the Army, the Department of Defense, or the U.S. Government.

U.S. Army War College
CARLISLE BARRACKS, PENNSYLVANIA 17013

ABSTRACT

AUTHOR: Lieutenant Colonel David K. Sarji

TITLE: Achilles' Head: Understanding the Challenges in Implementing Dependable and Secure Deeply Networked Military Embedded Systems

FORMAT: Civilian Research Project

DATE: 21 April 2008 WORD COUNT: 7,337 PAGES: 39

KEY TERMS: Embedded Systems, Security, Computing Systems

CLASSIFICATION: Unclassified

Embedded computing is an essential part of our military capacity. Experience with traditional information technology (IT) systems has made it abundantly that computing systems are subject to attacks, and that including security in the design process is a critical component in the development of new systems.

However, developing secure embedded systems is not a simple matter of transferring security techniques from traditional information technology environments. Due to the strenuous operational environment, frequent interaction with the physical world, and software application domain of embedded computer systems, traditional software and enterprise approaches to achieving dependability and security are only partially effective in securing embedded systems.

This paper will summarize basic concepts of traditional systems dependability and security, and their applicability to embedded systems. Constraints inherent in embedded systems will be discussed and threats to dependability and security summarized. The conceptual gap between regarding

security as merely the process of securing communication channels and developing truly secure, deeply networked, embedded systems resistant to all manner of attacks will be explored. Last will be a brief discussion of current best practices for attempting to achieve the appropriate level of dependability in embedded systems, up to and including ultra-reliable systems.

ACHILLES' HEAD: UNDERSTANDING THE CHALLENGES IN IMPLEMENTING DEPENDABLE AND SECURE DEEPLY NETWORKED MILITARY EMBEDDED SYSTEMS

1. INTRODUCTION

Embedded systems, the inclusion of digital processors in physical systems to deliver functionality, e.g. anti-lock brakes and fly-by-wire avionics, are fast approaching ubiquity. This trend will continue for the foreseeable future.

By postulating inclusion of embedded processing nodes in nearly all future military systems, various science fiction scenarios become easy to imagine, both good and bad. In the optimistic scenario, every vehicle, or even every individual soldier, is interconnected to a seamless control net that not only allows commanders perfect visibility and coordination of tactical movements on the battlefield, but also provides perfect communication at all levels of command, interfaces flawlessly with the Communications, Command, Control and Intelligence (C3I) systems of the other military services, and in its spare time performs a host of secondary functions like a perfect, just-in-time logistics ordering and delivery system. In the nightmare scenario, a terrorist or a well-funded adversary (up to and including a potentially hostile foreign government), gains control of a centralized control system and our missiles fall out of the sky, fighter planes spin out of control, gun systems point at the ground and lock, while all of our C3I networks crash simultaneously.

The reality, of course, is far more complicated.

Strategic Implications

According to the U.S. President's IT Advisory Committee, "We have become dangerously dependent on large software systems whose behavior is not well understood and which often fail in unpredictable ways."[1]

The reference speaks to software, but the point applies as well to embedded systems. We cannot simply ignore them or even limit their growth, because limiting embedded systems development would cause us to fall behind potential adversaries who successfully utilize (and protect) embedded systems. Our only option is to develop, secure and protect them.

In the coming decade, nearly all of our military systems, from top to bottom, from small to large, will be affected in one way or another by embedded systems. While not all delivered functionality is absolutely critical, military systems do differ from civilian applications in that even mid-level military systems can include functionality that could well mean the difference between life and death for a particular crew, or in aggregate might determine the outcome of a battle or even a war. Those are stakes high enough to qualify under any definition of life-critical systems, but the most deadly military systems force us to play for higher stakes still. A critical error in our nuclear weapon control systems still has the potential to cause the end of our entire species, and just

because we haven't done it so far doesn't mean that the danger has gone away. In fact, increasing systems complexity may have had the opposite effect, our trust in our systems increasing over the years because nothing has happened thus far, while our actual potential for errors and vulnerabilities may be increasing. According to Charles Perrow in his book, Normal Accidents, complacency and increasing complexity make accidents almost inevitible. [2]

We cannot avoid using embedded systems and many of them perform critical functions in our military systems. If they are compromised or fail our capabilities will be hugely degraded, perhaps fatally. To prevent this we must understand the strengths and weaknesses of embedded systems, understand the ways in which security in an embedded system differs from traditional enterprise information security, and include integrated security as a major design metric from the beginning of all embedded system development.

Embedded systems must never become our Achilles' Heel.

Target Audience

Many U.S. government and military acquisition executives have worked for years on Information Technology projects and are familiar with traditional approaches to system security. But it is important that the conceptual differences between enterprise security and embedded systems security is well understood, especially at a senior leadership level. Understanding the concepts that form the

foundation of embedded systems security and dependability will enable a senior executive to guide more unified design of the system. Unified design, including security built into the system from the beginning, is the only way to deliver a dependable, secure, embedded system.

There is no current government documentation that specifically addresses the requirements of embedded systems. The closest applicable documentation, MIL-STD-882D, provides broad requirements for system safety, by providing a framework of generalities and bureaucratic good practices, without delving into the specifics of any particular system type. It also puts signatory authority for accepting residual risk at an executive, rather than technical, level.

The intended audience for this paper is that executive, the senior acquisition leader who finds him or herself suddenly responsible for the security and dependability of embedded systems. Even years of prior software or I.T. experience are of limited utility in that, unfortunately, that experience does not transfer particularly well to the problems of embedded systems.

But that executive must nonetheless determine if residual risk is truly acceptable, and sign on the bottom line.

This paper is intended to impart a broad understanding of the ways in which embedded systems dependability and security differ from dependability and security for enterprise systems and may be considered an executive tutorial

on the broad topic of embedded systems dependability, with the goal of imparting a broad understanding of the concepts in order to allow a top level executive to influence the design process so as to much increase the likelihood of actually developing secure systems.

This paper will explain in broad and relatively understandable terms the current best-practice concepts and procedures that the civilian world is using to ensure dependability in embedded systems. In theory, military systems should not only incorporate these ideas, but should further refine them to reflect the additional threats to, and requirements of, military systems. In reality, most of our projects (as is the case with most civilian projects as well) probably do not utilize even the best civilian practices.

This paper is not intended to be an exhaustive or highly technical discussion of specific methodologies used in building highly dependable embedded systems. It is intended to be a synthesis of work that has gone before, a broad brush stroke picture understandable in one reading by an intelligent layperson, that will enable an executive to more effectively guide a design team to include proper dependability and security in the embedded systems design process.

A note on Absolute versus Relative Dependability

Under the current processes by which integrated embedded systems of hardware and software are produced, no absolute, yes/no, guarantee of dependability is possible. This fact applies as fully to military systems as to civilian systems. Very reliable or even ultra-reliable systems can be built, but the more reliable a system becomes, the more impossible it becomes to test its extended reliability. There is no way to quantify the outer limits of ultra-reliable systems, because even exorbitantly funded testing can never achieve a guarantee of absolute reliability.[3]

Hardware can also be expected to fail, but it is at least easier to test. However, life testing – the main reliability evaluation technique for individual electronic components[4] - cannot be applied to software. The prediction of software reliability thus has one less significant tool at its disposal than does hardware reliability prediction.

Software or hardware, the best reliability that can be achieved is only a very high probability that no failures will occur. Perfection is impossible to achieve. This remains true because of a variety of factors, such as the ever increasing complexity of software, the inability to perfectly control compilers and development tools, programming-language vulnerabilities, or the danger of

intentionally programmed back doors or Trojans, which can never be entirely discounted.[5]

No amount of signed forms or accreditations ever changes this fact.

No system is ever perfectly reliable.

2. DEFINITIONS

According to Wikipedia, "An **embedded system** is a special-purpose computer system designed to perform one or a few dedicated functions, often with real-time computing constraints. It is usually embedded as part of a complete device including hardware and mechanical parts."[6]

Dr. Philip Koopman of Carnegie Mellon University offers a definition of a new type of system that is currently coming into existence: **"Deeply networked systems** are formed when embedded computing subsystems gain connectivity to each other and to larger enterprise systems."[7]

The original definition of **Dependability** was the ability to deliver service that could justifiably be trusted. That definition was an attempt to summarize the attributes of dependability into single phrase. However, "As developed over the past three decades, dependability is an integrated concept that encompasses the following attributes:

- **availability**: readiness for correct service
- **reliability**: continuity of correct service

- **safety**: absence of catastrophic consequences on the user(s) and the environment
- **integrity**: absence of improper system modifications
- **maintainability**: ability to undergo modifications and repairs."[8]

Security is a composite of the dependability attributes of integrity and availability, with the addition of confidentiality.[9]

3. UTILITY OF TRADITIONAL ENTERPRISE SECURITY TECHNIQUES

Some techniques used in traditional enterprise security are also of use in embedded systems security. Chief among them is the use of encryption to secure network communications, which is already frequently used with embedded systems. Firewalls are also important, especially at the interface between the enterprise and distributed systems, where firewalls must be installed on both sides of the communication channel. The firewalls for embedded systems are not precisely the same as firewalls for enterprise systems, however, and while work is under way at this time to develop appropriate firewalls for this interface, it has not yet been completed.

Other techniques which work well for enterprise systems security and dependability do not, in fact, transfer well to embedded systems. This is because the constraints and application spaces differ tremendously.

Enterprise systems tend to be transactional, centered around requests, updates to a central data repository, and responses. Servers are housed in a large, climate controlled facility and users typically access the system via a desktop computer. The nature of the applications makes possible certain strategies to provide fault tolerance and retain system functionality, such as "roll back" recovery, in which a system that has experienced a fault is returned to a previous correct state (checkpoint), from which all changes are reapplied. This strategy works well, for example, in banking or personnel applications.

But embedded systems are out in the real world, operating in real-time, frequently interacting with the physical world in a quick control loop. Embedded systems contain much less state (a unique configuration of a machine, that may be thought of as analogous to system settings) than enterprise systems, and tend to focus on extracting data from the system in real time, using it and quickly discarding it as it becomes stale. In an embedded system controlling physical actuators it is likely to be impossible to roll back a state change – when faced with a fault the system must instead "roll forward" and continue operation uninterrupted. This can be accomplished in different ways, but the point is that the system can't ever stop. It can't ever go back, so a whole strategy for recovery is closed to it.

Roll forward recovery versus roll back recovery is one example of the ways in which embedded systems differ from traditional enterprise IT applications. Other examples include the small size and weight requirements for embedded processors (as opposed to the relatively unlimited size, weight, and power draw of enterprise servers or desktops), continuous operation of single nodes often using battery power, use of embedded network protocols with less inherent capability than TCP/IP (which itself contains many vulnerabilities), and no one being a system administrator for all of the innumerable individual embedded processors, e.g. no one is a sysadmin for a toaster, even if it is connected to the internet.[10]

4. CONSTRAINTS ON EMBEDDED SYSTEMS

Embedded systems suffer from constraints not experienced by traditional computer systems. This is due to several factors inherent in distributing processing power to many small nodes physically embedded in hardware.

Price

Civilian systems are constrained by the power of the marketplace, which rewards the lowest possible price. Even a small difference in per-unit cost for distributed processors can make a very significant difference when building millions of units per year. The amount spent per node must be held to an absolute minimum in civilian systems, in order to achieve price competitiveness.

This means that each node is typically implemented using the cheapest possible processor, with the minimum processing power necessary to perform the functionality assigned. Implementing security measures that run in each distributed node requires local processing power to run the security algorithms. Because of the cost associated with providing additional processing power, many civilian embedded systems are implemented with processors that are not capable of handling security processing in addition to the load imposed by their primary functional task.[11]

Battery life

Embedded systems often have significant constraints on their energy usage, and many are battery powered. Some embedded systems are able to recharge daily, but others must operate for long periods of time on batteries alone. Additional processing, such as for security functions, requires additional power. Use of battery power also creates vulnerability if there is any way to force the embedded system to operate in power-hungry mode (e.g. wireless communications) to the point of battery failure. An attacker may fail to break into a system, but succeed in bringing it down by draining the battery.[12]

Environment

The environments in which embedded systems operate are often harsh. Embedded systems are expected to operate in the real world, under extremes of

heat, cold and physical pounding to which traditional computer systems are not subject. This is true of both military and civilian systems.

However, conditions likely to be encountered in military use environment must be viewed as considerably more forbidding than anything faced by civilian systems. Not only is the battlefield the harshest possible physical environment, but military systems must also be able to withstand a multitude of attacks, not merely from internet hackers and criminals, but also by extremely large, well funded and organized attempts to compromise them, such as might be mounted by any number of technologically adept nation states.[13]

Processing power cost should not be a problem for military systems

Military systems suffer from the same size, battery and environmental constraints as civilian systems, but have one advantage in that the military is not subject to competitive price pressures. Cost should not be a significant factor in purchasing slightly more processing power per processor, if necessary to implement proper security measures.

Which is only one small step in the process. It by no means implies success in actual implementation of secure embedded systems. But at least in this instance the military appears to face one less constraint to developing secure systems.

It can be argued that spending money in any area reduces spending in other areas, that there are only so many slices of the pie to go around and that the overall process is a zero sum game. However, common sense dictates that spending an additional few cents per unit in multi-million dollar systems is a wise allocation of resources.

5. FAULTS: THREATS TO DEPENDABILITY

Faults can be intentionally or accidentally created, but any fault, whatever its origin, that causes a disruption in proper function or service is a threat to dependability. Threats to the dependability of a system can occur during either part of the life cycle of a system, the development phase or the use phase.

All faults that can affect a system belong to three major partially overlapping groupings:

- **development faults**: that include all fault classes occurring during development
- **physical faults**: that include all fault classes that affect hardware
- **interaction faults**: that include all external faults, such as operator mistakes and malicious entities[14]

During the development phase, development faults can be introduced into a system by its development environment. Potential sources include:

1. The physical world

2. Human developers, some of whom may be incompetent or malicious
3. Development tools: software and hardware used by the developers to assist them in the development process
4. Production and test facilities[15]

During the use phase the system interacts with its use environment. The use environment presents a different set of partially overlapping potential sources of fault introduction:

1. The physical world
2. Administrators, some of whom may be incompetent or malicious
3. Users
4. Providers: entities that deliver services to the system at interfaces
5. Infrastructure
6. Intruders: malicious entities (humans and other systems) that attempt to gain control of or access to a system, alter service or halt it.[16]

Clearly there are many opportunities for the introduction of faults in the life of every system.

6. THREATS TO SECURITY

One source of fault introduction is the malicious entity, the theoretical attacker using decryption to steal our passwords. So we encrypt our communications.

But real "...{attackers} rarely take on the theoretical strength of well-designed cryptographic algorithms. Instead, they rely on exploiting security vulnerabilities in the software and hardware components of the implementation."[17]

Any means which allows fault introduction, whether access is gained through software or hardware or any combination of the two, is a threat to dependability.

Physical and side-channel attacks

"Physical and Side-channel Attacks are typically classified into invasive and non-invasive attacks. Invasive attacks involve gaining physical access to the appliance to observe, manipulate, and interfere with the system internals. Since invasive attacks against integrated circuits typically require expensive equipment, they are relatively hard to mount and repeat."[18]

Physical attacks can include de-packaging of a system chip. This is accomplished by dissolving the chip's outer protective layer of resin by using acid. A combination of microscopy and continued invasive stripping away of layers can reveal the internals of a chip. This, of course, requires extensive effort and equipment. However, knowledge of the internal layout of a chip can help in the design of a software or side channel attack to be used later.

Side channel attacks include timing analysis and power analysis. In timing analysis, a system's cryptographic keys can be determined by analyzing small variations in the time required to perform cryptographic communications. In power analysis, the power profile of cryptographic computations can be used to calculate the cryptographic key used. Both timing and power analysis illustrate ways in which devices leak information, even if communications are protected via cryptography.[19]

Not considered to be as viable an attack form against embedded systems, **electromagnetic analysis attacks** can be used to reconstruct the screen contents of a video display unit, or could be attempted against chips themselves to reveal sensitive information. But attacks against individual chips would be hard to mount, as they would require intimate knowledge of the chip's layout, and monitoring equipment placed so as to detect the tiny electromagnetic fluctuations emitted by a processor.

Factors working against secure software

Three factors work together to make managing security in software difficult:

- **complexity**: more lines of code make bugs and vulnerabilities more likely. As embedded systems communicate with each other and

enterprise systems, to become deeply networked systems, the associated software grows more complex.

- **extensibility**: modern systems are designed to be extensible, to allow updates or extensions to increase or evolve functionality. Extensible systems make it hard to prevent software vulnerabilities from slipping in.
- **connectivity**: connectivity makes it possible for small breaches to cascade into large ones. Attackers no longer need to gain physical access to a system to attack it.[20]

Software attacks

If communication with an embedded device is possible, software attacks such as rootkits can be carried out against the operating system kernel of an embedded processor. If access is gained, the kernel has the ability to write to BIOS memory. In the past, the BIOS was stored in chips that could not be updated, but modern embedded systems often utilize chips with firmware that can be rewritten from software, i.e. flash ROM.

Flash ROM is almost never fully utilized and leaves plenty of room to install malicious software. Once installed, it is immune to system reboots and operating system re-installation. This software is called a hardware virus.

"A simple hardware virus may be designed to impart false data to a system, or to cause the system to ignore certain events. Imagine an anti-aircraft radar that uses an embedded real-time operating system. Within the system are several Flash ROM chips. A virus is installed into one of these chips and it has trusted access to the entire bus. The virus has only one purpose – to cause the radar to ignore certain types of radar signatures."[21]

Threat implications for military systems

The military systems threat environment overlaps the civilian threat environment but is not precisely the same. Any device that contains the ability to communicate with the outside world can expect to be attacked, whether military or civilian.

However, as the military moves to deeply networked systems of sensors and vehicles, it is unlikely that we will hand out examples of our embedded systems to the public at large, as is the case with many civilian embedded systems, such as those in phones and PDAs. We are thus not as prone to having our systems pried apart, analyzed and cracked by random hobbyists.

This is not an unadulterated good. In many ways, public exposure is the acid test for system security. But military systems face a catch-22; they will never be distributed to the public, but this deprives the military of the use of the public as a resource to improve the security of our systems.

And we face significant threats that may never even appear unless we have a serious confrontation with another technologically adept world power. This would bring about a situation with the potential to unfold in a radically different manner than confrontation with non-technological entities. This is, of course, not the most likely threat. But it is the one with the most serious consequences.

It has already been discussed how faults may be introduced in many different ways throughout the system lifecycle. The possibility of espionage and intentional fault introduction in our systems cannot be overlooked. While there is some attempt made to vet programmers before allowing them to work on critical systems, common sense dictates that among the vast numbers of foreign programmers and system developers working on our systems, a small number are actual intelligence assets of foreign powers, US secret clearance or not. And a greater number would have more loyalty for their home country than for the US.

This is not meant to disparage the vast majority of foreign nationals who do excellent work for us. But to pretend that major players such as China, India, and Russia do not have visibility into our systems would be to bury our heads in the sand. Even if the numbers of actual back doors or Trojans left in programs is very small, the intimate knowledge of our systems (and perhaps even literal copies of the associated software) would allow much more sophisticated and

effective attacks to be launched. And, of course, any major nation-state has the ability to organize and fund development efforts far in excess of any private group

We will never know what weapons lie in the vaults of foreign powers, waiting to attack our systems. All we can do is build the most secure systems possible, with layered software security and tamper proof physical implementation, and hope that nothing gets through.

7. TRADITIONAL MEANS TO ATTAIN DEPENDABILITY AND SECURITY

"The design of high availability systems is based on a combination of redundant hardware components and software to manage fault detection and correction without human intervention."[22]

While the specific techniques used to achieve high dependability in enterprise systems differ significantly from those used in embedded systems, the higher level concepts are applicable to both traditional systems and embedded systems. Several concepts have been developed and implemented over the years to attain dependability and security in systems. They address the concept of faults and how to deal with them throughout the lifecycles of the system, and can be grouped into four major categories:

- **Fault prevention** : to prevent the occurrence or introduction of faults

- **Fault tolerance**: to avoid service failures in the presence of faults
- **Fault removal**: to reduce the number and severity of faults
- **Fault forecasting**: to estimate the present number, the future incidence, and the likely consequences of faults.[23]

Fault prevention and fault tolerance are the means by which a system is made dependable enough to be trusted. Fault removal and fault forecasting serve to justify that confidence, showing that, "…the functional and the dependability and security specifications are adequate and that the system is likely to meet them."[24]

In other words, fault prevention and tolerance keep the system running, fault removal increases confidence in the software by removing faults, and forecasting attempts to predict future faults (or lack thereof) so as to give confidence that the system will function as designed.

However, no amount of theory changes the fact that if security is not included as a design metric from the beginning of system development, it is unlikely to be successfully achieved later in the process. As per *Security as a New Dimension in Embedded System Design*, "…software security must be part of a full lifecycle approach. Just as you cannot test quality into a piece of software, you can not spray paint security features onto a design and expect it to become secure."[25]

The single most effective step that can be taken to improve systems security is to include it in the development process from the beginning.

Fault prevention

Fault prevention attempts to prevent introduction of faults during the development and use (maintenance) phases of a system lifecycle. It includes attempts to address both malicious and accidental faults.

Software best practices meant to produce high quality software are a major component of fault prevention during development and should be used whenever possible, though they are too extensive to discuss here.

Fault prevention also uses analysis techniques to identify potential failures with severe consequences and lower the probability of their occurrence.

Analysis techniques to determine failures for remediation commonly include:

- **Failure Mode and Effects Analysis** (FMEA), which looks for consequences of component failures, but requires expert knowledge of the system to decide what to analyze.
- **Failure Mode and Effects Criticality Analysis** (FMECA), which adds two columns to FMEA, an overall assessment of criticality and possible actions to reduce criticality.

- **Fault Tree Analysis** (FTA), which eliminates single-point vulnerabilities by protecting them with an "AND" gate. FTA utilizes backward chaining and analyzes possible hazard origin, although a pre-generated list of hazards is necessary for analysis.[26]

Fault tolerance

Faults can never be completely eliminated. Systems must tolerate them and continue to function.

Fault tolerance is implemented by using redundant or diverse implementation of a system to avoid the effects of faults. The main concept utilized is **redundancy**, which can come in a variety of forms:

- Hardware redundancy
- Software redundancy
- Time redundancy
- Information redundancy[27]

"**Hardware redundancy**. Rather than building hardware as individual super-reliable modules constructed of super-reliable components, it is usually more cost effective to use redundant replicated modules of everyday commercial quality hardware…"[28] Multiple physical systems can provide excellent fault tolerance for a relatively low price.

Software redundancy has become somewhat discredited. It was traditionally attempted by means of N-version programming, in which N versions of software with identical functionality were produced by N separate teams. This was meant to produce identically-functioning software that did not contain the same fault. However, the price of such duplicate development was unacceptably high[29] and recent research has shown it to be of questionable utility at best[30], due to several factors, including similar algorithms, the tendency of a second team of programmers to be less competent than a primary team, and the existence of a completely different set of faults in the alternate software package. Current practice is to spend all of the time and money available to produce the best-quality single software solution that can be produced, using the best available infrastructure, software development tools, techniques and testing.[31] There are variations on software redundancy that are still useful, such as using two versions of software, one simple and one complex, to crosscheck solutions and provide gracefully degraded capability if necessary.

Time redundancy typically involves checkpointing, taking a "snapshot" of a system as it enters a new transaction. The kind of error recovery made possible by this approach is backward error recovery, or roll back recovery, which is not typically an option for embedded systems.

Information redundancy is typically accomplished by retaining multiple copies of information in separate locations, which also allows for backward error recovery, and is not typical of embedded systems.

Forward error recovery is a more common fault tolerance methodology for embedded systems. The main concept underlying forward error recovery is to continue from an error state and make corrections that will enable the system to resume normal operation.

As per Design Patterns for High Availability:

"One design pattern for forward error recovery is called alternative processing. This approach can be used when there are two (or more) processing alternatives for a transaction, with one alternative normally being very precise but computationally complex, and the other alternative being more simplistic and of higher performance."[32]

The two processes are compared, with the simple process used as both a test and a yardstick against the complex process. If they do not match, the simpler processing solution is used, which illustrates a final concept in fault tolerance known as **graceful degradation**, which allows sensible partial functionality to continue if full system performance cannot be provided. Graceful degradation can be provided by utilization of simpler algorithms, mechanical backup, or manual operation by a human being. A system that

degrades into a safe state with limited capability is sometimes said to be in "limp home" mode.

Fault removal

Fault removal takes place during both the development phase and the use phase of a system.

Fault removal during the development phase consists of three steps: verification, diagnosis, and correction. The basic concept is that the system is checked for its adherence to given properties, called verification conditions. If it does not the error must be located and remediated, and the verification conditions checked again.

Fault removal during use is corrective or preventive maintenance. Corrective maintenance seeks to repair reported faults in the system, while preventive maintenance attempts to identify and correct faults before they occur in the actual operational system.[33]

Fault forecasting

"Fault forecasting is conducted by performing an evaluation of the system behavior with respect to fault occurrence or activation."[34]

Typical fault forecasting involves constructing a **reliability model** of the system, which is typically one of three types:

- **parts-count models** that offer a conservative measurement based on the idea that a failure of any component will cause system failure;
- **combinatorial models**, which are extensions of parts-count models
- **state-space models**, wherein each possible combination of "up" or "down" components is represented by a corresponding state in the model, allowing for estimation of the system's probability of being in each state.[35]

Attempts to utilize fault forecasting in embedded systems development have met with mixed success. It works well up to the ultra-reliable level, but research thus far indicates a near complete inability to quantify reliability in life-critical, ultra-reliable software. Classical quantification techniques transferred to life-critical systems have resulted in immense escalation of testing costs without actually being able to guarantee system reliability.[36]

8. DEPENDABILITY AND SECURITY STRATEGIES FOR EMBEDDED SYSTEMS

Embedded systems lack built-in security

Because embedded systems are often used in life-critical systems (e.g. anti-lock brakes, avionics, etc) there has been a good deal of work in creating dependable embedded systems. This work has included hardware fault tolerance, software fault tolerance, and organized techniques to develop high

quality software. However, previous generations of embedded systems used as an assumption that they were closed systems, inaccessible from the outside. As a result, traditional embedded systems have little or no native security.[37]

Dependability design requirements for embedded systems

Different implementations of embedded systems will drive the dependability measures needed. To achieve dependability, embedded system engineering must take into account the need to prevent:

1. Physical damage to the system
2. The unauthorized release of information from the system
3. The inability to deliver service from the system (e.g. denial of service attacks)[38]

Best practices in developing dependable embedded systems

Many methods already exist to deliver dependable embedded systems. The techniques are often interrelated and must be integrated as part of a unified design.

Traditional methods of fault prevention, fault tolerance, and fault removal are all applicable to embedded systems. In addition, there are some relatively inexpensive but effective methods to improve embedded security that should be included by default in modern systems. These "best practices" address at least

some of the challenges posed to the security and dependability of embedded systems.

- **Trusted Time Base** provides a foundation for resolving timing disruptions and ambiguities, such as distinguishing whether a tightly spaced group of messages is due to congestion or attack, detecting timing jitter, and compensating for message aging in closing internal control loops.
- **Watchdog timers** installed in the embedded systems processors, to resolve timing issues and protect against attack
- **Firewall protection in both directions** between enterprise systems and embedded systems. It is important to have a firewall to protect the embedded system from external attack, but it is just as important to have a firewall to protect the enterprise from intrusion and exploitation from a compromised embedded system, or from use of a communication channel intended to be used by an embedded system.
- **Limiting damage from compromised servers.** The amount of control that each individual server can exert over the entire universe of physical sensors and actuators should be limited – i.e. the compromise of one server should never lead to loss of control of

a vast fleet of subsystems. One potential approach would be to limit the number of embedded systems that are allowed to take direction from a particular server, even if those directions are routed by indirect paths.[39]

Fault forecasting is meaningless for ultra-reliable systems

Fault forecasting can be applied to embedded systems. Reliability models function sufficiently well to ensure dependability to a relatively high degree of reliability.

However, many embedded systems require ultra-reliability. Many different dependability techniques have been used and combined to achieve extremely high levels of reliability. However, quantifying this reliability has proven to be problematic. Demonstrating ultra-reliability requires an inordinate amount of testing, and even at the end of the testing process there is no real guarantee that the system is truly ultra-reliable; the air is simply too rarefied for traditional analysis tools to do the job.

This is considered an open research problem at this time.[40]

All future military embedded systems should include security features at the individual node level

Even in the near future, many military embedded systems applications will continue to resemble today's anti-lock brake systems, in that there will continue to be no direct communication with the outside world. Multiple

systems of sensors in a vehicle (e.g. airplane, helicopter, or armored vehicle) will communicate primarily along internal channels to a single "brain" that collates all information and is the sole point of communication to the enterprise network. Even though that brain and its communication channels may in fact include redundant systems to improve dependability, it can be logically regarded as a single access point into the system.

Much embedded systems security work to this point has been focused on encryption of the network traffic passing back and forth between the embedded system and the enterprise. This work has led to efforts to develop specialized firewalls operating in both directions between the enterprise and the embedded system.

This, however, is not enough to consider in the design of future systems. There are two reasons.

First is that the capabilities that become possible with embedded systems communicating directly and individually to the outside world cannot be ignored. Even if they are not initially included, they will creep into designs, as users learn about and request additional functionality. It is better to plan ahead for individual distributed nodes to have communication capabilities, and institute the proper practices that will ensure security is part of every embedded system node design, than to wait and attempt to secure nodes retroactively.

Second is that even in the "single" communication channel scenario, there remains a need for security for the systems located behind the firewall, defense in depth, so that a penetration of the firewall does not result in total compromise of the system.

9. SAFETY CRITICAL ENGINEERING

Much of the work in dependability is based on an evaluation of the severity of the potential fault, from crippling the system completely to being only of nuisance value. The more significant the effects of a fault, the lower the likelihood of its occurrence must be.

Work in aircraft systems is probably the closest civilian analogue to military systems, because aircraft systems are, of course, life-critical. The generally acceptable thresholds of frequency of occurrence that apply to aviation systems can be applied to military life-critical systems with a fair degree of confidence.

Civil Aircraft Hazard Categories

- Catastrophic (10^{-9}/hr)
- Hazardous (10^{-7}/hr)
- Major (10^{-5}/hr)
- Minor (10^{-3}/hr)
- Nuisance (10^{-2}/hr)[41]

Note, however, that the penalty associated with these thresholds is life critical but relatively limited in scope, e.g. loss of a fully loaded airliner is a tragedy, but it is not literally the end of the world.

For systems that very well might cause the end of the world, were they to malfunction, an even higher level of dependability would probably be in order. But there is no way to test such dependability. In fact, we cannot even truly confirm dependability in the 10^{-9} range.

Safety Integrity Levels

Safety Integrity Levels (SILs) assign each level of risk an acceptable likelihood of occurrence. The level of integrity required varies according to the level of risk involved in a fault, with high levels of integrity required for high risk and lower levels acceptable for low risk.[42]

There are five SILs that determine how critical a system or component failure would be, and they dictate the acceptable frequency of occurrence. They are ranked according to possibility of safe recovery from a fault, and if multiple faults have different levels of controllability, the highest single controllability risk determines the SIL.

SIL	Controllability	Acceptable Failure Rate
4	Uncontrollable	Extremely Improbable (10^{-9}/hr)
3	Difficult to control	Very Remote (10^{-7}/hr)
2	Debilitating	Remote (10^{-5}/hr)
1	Distracting	Unlikely (10^{-3}/hr)
0	Nuisance only	Reasonably Possible (10^{-2}/hr)

10^{-2}/hr = .01 occurrences per hour = 100 hours of operation per failure

10^{-3}/hr = .001 occurrences per hour = 1000 hours of operation per failure

10^{-5}/hr = .00001 occurrences per hour = 100,000 hours of operation per failure

10^{-7}/hr = .0000001 occurrences per hour = 10,000,000 hours of operation per failure

10^{-9}/hr = .000000001 occurrences per hour = 1,000,000,000 hours of operation per failure.

1,000,000,000 hours = 41,666,667 days = 114,155 years

That's a long time between failures. And it would be nice to think that it was possible to achieve that level of reliability, and prove it. Because what the numbers really mean is that a high standard has been set, that a mathematical probability solution has been offered for how often we think a particular thing should happen. And that we'd like disastrous software faults to happen so rarely that they never actually happen at all, and so have set the bar very high.

But a theoretical definition of how rarely something should occur is only a definition, not a reality. It does not mean that any of our systems are actually that safe, or that there is any way to prove whether they are or not, out at the fringes of ultra-reliability.

Each SIL is associated with specific techniques that should be utilized during development and use. These include specification and design processes, specific languages and compilers, configuration management products, configuration management processes, testing, verification and validation, and access for assessment.[43]

SIL techniques are mainly focused on software engineering. Some of the techniques associated with SILs are thus software engineering techniques that may not entirely apply to embedded systems, though there is utility in considering every requirement associated with a particular SIL and its potential application to an embedded system. SIL recommendations are a good starting point. It is important to remember that in real life, even SIL 3 is very hard to achieve, and SIL 4 would require multiple Ph.D. level specialists in safety critical software to be involved in the project.[44]

Two candidate embedded safety critical development processes

There are two development processes currently utilized in safety critical engineering that may assist in development of a dependable and secure embedded system.

- MISRA – Motor Industry Software Reliability Association
  - Automotive specific
- IEC 61508
  - Newer, more generic, broad acceptance in non-automotive industry[45]

10. MIL-STD-882D

MIL-STD-882D is the Department of Defense document which guides Standard Practice for System Safety. This is the document which comes closest to governing the development of critical embedded systems, as it requires analysis of fault likelihoods and impacts, and provides guidance for an acceptable likelihood of occurrence for system faults, based on the severity of damage in case of a failure of that type.

"The objective of system safety is to achieve acceptable mishap risk through a systematic approach of hazard analysis, risk assessment and risk management."[46]

MIL-STD-882D contains the following general requirements:

- Documentation of the system safety approach
- Identification of hazards
- Assessment of mishap risk
- Identification of mishap risk mitigation measures
- Eliminate hazards through design selection
- Incorporate safety devices
- Provide warning devices
- Develop procedures and training
- Reduction of mishap risk to an acceptable level
- Verification of mishap risk reduction
- Review of hazards and acceptance of residual risk by the appropriate authority
- Tracking of hazards, their closures, and residual mishap risk[47]

Appendix A to the MIL-STD provides guidance for the implementation of safety engineering in DoD acquisitions. The primary means for addressing system safety is based on a multi-step process:

- The mishap is categorized according to the severity of impact if it occurs, in descending order from catastrophic to negligible
- The mishap is categorized according to the probability of its occurrence, in descending order from frequent to improbable

- The severity and probability are cross indexed to provide a risk assessment value, which places the fault into a particular risk category, in descending order from high to low
- The level of signatory authority required to accept risk is correlated with the risk category, with higher level authority required to accept higher level risk[48]

MIL-STD-882D is a useful document, but it is intentionally nonspecific. The requirements address general risk management process but provide no specific guidance for the development of secure embedded systems. Developers will have to adhere to its guidance, but secure systems development requires application of far more specific concepts and techniques than those contained in 882D in order to be effective.

11. CONCLUSIONS

Embedded systems are an inescapable part of the future, and require dependability strategies tailored to their unique requirements and situation. Some low cost/high return security features should be incorporated in all future embedded applications, and additional methods to increase dependability should be applied as the criticality of the component or system increases.

Traditional methods for achieving dependability are often useful in achieving embedded systems dependability and should be considered a baseline

to which additional techniques may be added as the requirement for dependability increases. But the differences inherent in embedded systems: the operational environment, application space, and frequent interaction with the physical world, must also be understood.

Threats against military systems are difficult to define precisely, but it is certain that every attack which might be used against a civilian system serves only as a starting point in studying the attacks that might be used against military targets. Above all, security must be considered from the first day of embedded system design. It must be integrated in the design from the start, and address all potential avenues of attack, be they physical, side channel, or software.

12. REFERENCES

[1] US President's IT Advisory Committee, February 1999.

[2] Charles Perrow, *Normal Accidents: Living with High-Risk Technologies*, by Charles Perrow, Basic Books, NY, 1984.

[3] Ricky W. Butler and George B. Finelli, "The Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software," IEEE Transactions on Software Engineering, Vol. 19, No. 1 (January 1993): p.3.

[4] Andrew L. Reibman and Malathi Veeraraghavan, "Reliability Modeling: An Overview for System Designers," IEEE COMPUTER (April 1991): p. 49.

[5] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing," IEEE Transactions of Dependable and Secure Computing, Vol. 1, No. 1 (January-March 2004), p.14

[6] "Embedded System," Wikipedia, http://en.wikipedia.org/wiki/Embedded_system, 2008.

[7] Philip Koopman, Jennifer Morris, Priya Narasimhan, "Challenges in Deeply Networked System Survivability," NATO Workshop: Embedded Security (August 2005).

[8] Avizienis, “Basic Concepts and Taxonomy of Dependable and Secure Computing,” p.13.

[9] Ibid.

[10] Philip Koopman, “Challenges in Deeply Networked System Survivability,” p.3.

[11] Philip Koopman, “Embedded System Security,” Computer, July 2004, p.95.

[12] Ibid, p.95-96.

[13] Ibid, p.95.

[14] Avizienis, “Basic Concepts and Taxonomy of Dependable and Secure Computing,” p. 15.

[15] Ibid.

[16] Ibid, p.14

[17] Paul Kocher, Ruby Lee, Gary McGraw, Anand Raghunathan, Srivaths Ravi, “Security as a New Dimension in Embedded System Design,” Annual ACM IEEE Design Automation Conference, Proceedings of the 41st annual conference on Design automation (June 7-11, 2004): p.755.

[18] Ibid, p.756.

[19] Ibid.

[20] Ibid, p.755.

[21] Ibid, p.755.

[22] David Kalinsky, “Design Patterns for High Availability,” Embedded Systems Programming (August 2004): p.24.

[23] Avizienis, “Basic Concepts and Taxonomy of Dependable and Secure Computing,” p.14.

[24] Ibid.

[25] Kocher, “Security as a New Dimension in Embedded System Design,” p. 755.

[26] Philip Koopman, “Critical Systems and Software Safety,” Class 18-649 “Distributed Embedded Systems,” lecture 18 briefing slides, Carnegie-Mellon University, March 29, 2007, p.9.

[27] Kalinsky, “Design Patterns for High Availability,” p.25.

[28] Ibid.

[29] Ibid, p.28.

[30] Butler, “The Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software,” p.3.

[31] Kalinsky, “Design Patterns for High Availability,” p.28.

[32] Ibid, p.32.

[33] Azizienis, “Basic Concepts and Taxonomy of Dependable and Secure Computing,” p.27, 28.

[34] Ibid, p.28.

[35] Reibman, “Reliability Modeling: An Overview for System Designer,” p. 51.

[36] Butler, “The Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software,” p.3.

[37] Koopman, “Challenges in Deeply Networked System Survivability,” p.2.

[38] Koopman lecture 18, p.?

[39] Challenges in…, p.6

[40] The Infeasibility of Quantifying the Reliability, p.1

[41] Koopman, “Critical Systems and Software Safety,” briefing slides, p.23.

[42] Philip Koopman, “Critical Systems Engineering,” Class 18-649 “Distributed Embedded Systems,” lecture 21, briefing slides, Carnegie-Mellon University, April 10, 2007, p.14, 15.

[43] Ibid, p.20-22.

[44] Ibid, p.23.

[45] Ibid, p.7.

[46] U.S. Department of Defense, *Standard Practice for System Safety*, MIL-STD-882D, (Washington, D.C.: U.S. Department of Defense, 10 February 2000), p.3.

[47] Ibid, p.3, 4.

[48] Ibid, Appendix A, p.18-20.